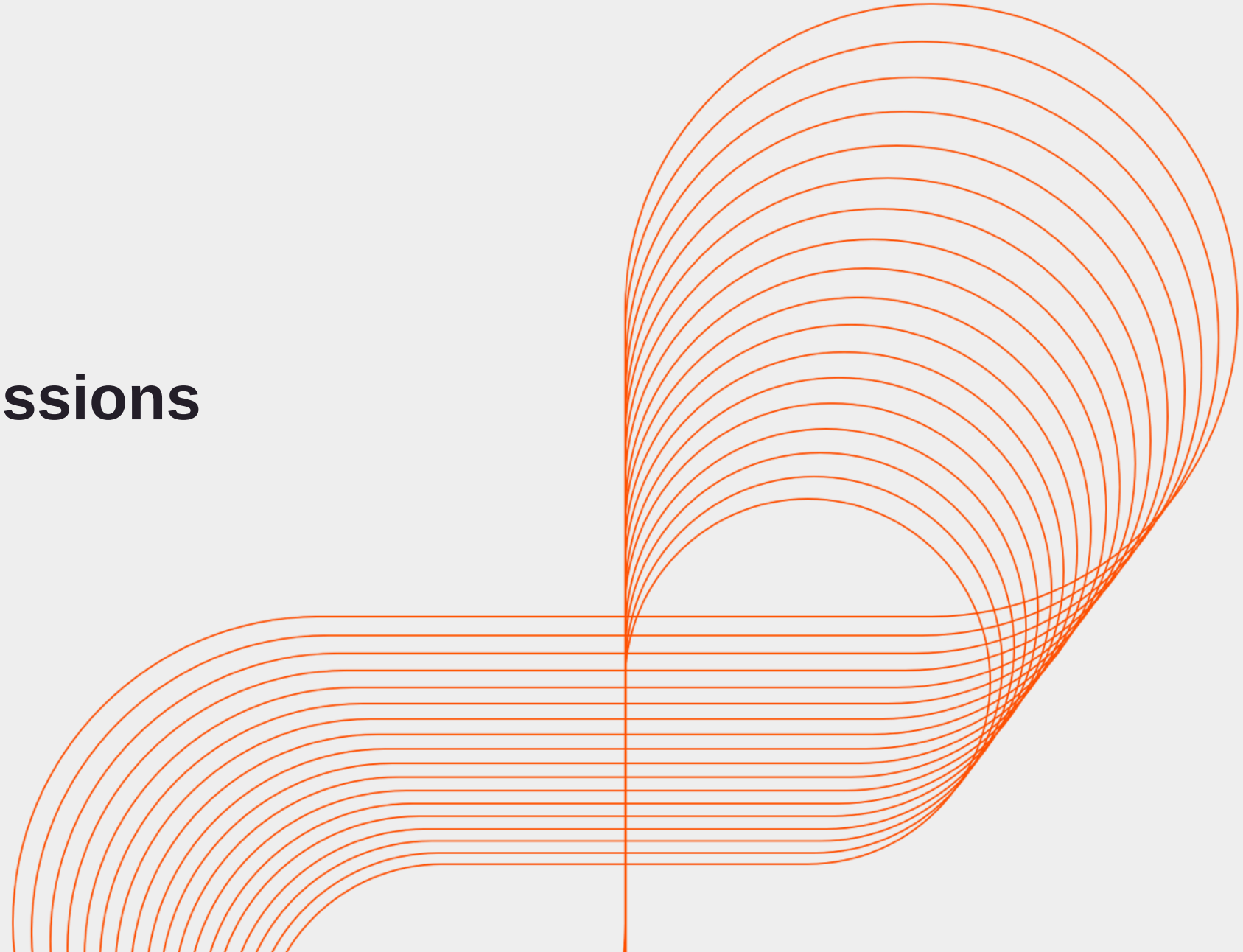
Persistent

Core Java:
Regular Expressions

Persistent University



Keep learning points:

- Understand regular expressions
- Understand and use Pattern and Matcher classes
- Classes along with important methods like compile, match, find, start, end, group etc.
- Creation of regular expressions, important symbols, meta characters etc.

What is Regular Expression?

- A regular expression is a special sequence of characters that helps to match or find other strings or sets of strings, using a specialized syntax held in a pattern.
- Can be used to search, edit, or manipulate text and data.
- It is widely used to define constraint on strings such as password and email validation.
- The **java.util.regex** package has classes for using Regular Expressions.

The java.util.regex package

- The **java.util.regex** package consist of following classes.
  - Pattern class : used to specify regular expression.
  - Matcher class : used to match the pattern against another character sequence.

The Pattern class

- A **Pattern** object is a compiled representation of a regular expression.
- Represents the pattern a string must follow.
- No public constructors so object can not be created.
- The compile method is used to retrieve the pattern object.
- This method requires a parameter which is a regular expression.

```
Pattern pattern = Pattern.compile(//regular expression);
```

The Matcher class

- **Matcher** object performs operations related to regular expression.
- No constructor available so object to be created by using factory method of Pattern class.

```
Matcher matcher = pattern.matcher(CharSequence str);
```

- **CharSequence** is the interface that defines read-only set of characters.
- String class implements this interface.

Demo : create Pattern & Matcher objects

- This will create object of pattern with the regular expression. Matcher object can be used to perform further operations by calling methods.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegularExpressionDemo1 {
  public static void main(String[] args) {
            Pattern pattern = Pattern.compile("Java");
            Matcher matcher = pattern.matcher("Java");
  }
}
```

Matcher class methods

- There are various categories of methods available in Matcher class which are as follows:
  - Index methods : provide useful index values that show where the match was found in the input string
  - Study methods : review the input string and return a Boolean value that indicates whether pattern is found or not
  - Replacement methods : useful methods for replacing text in an input string

Index Methods

Method	Description
public int start()	Returns the start index of the previous match.
public int start(int group)	Returns the start index of the subsequence captured by the given group during the previous match operation.
public int end()	Returns the offset after the last character matched.
public int end(int group)	Returns the offset after the last character of the subsequence captured by the given group during the previous match operation.

Study Methods

Method	Description
public boolean lookingAt()	Attempts to match the input sequence, starting at the beginning of the region, against the pattern.
public boolean find()	Attempts to find the next subsequence of the input sequence that matches the pattern.
public boolean find(int start)	Resets this matcher and then attempts to find the next subsequence of the input sequence that matches the pattern, starting at the specified index.
public boolean matches()	Attempts to match the entire region against the pattern.

Replacement Methods

Method	Description
public String replaceAll(String replacement)	Replaces every subsequence of the input sequence that matches the pattern with the given replacement string.
public String replaceFirst(String replacement)	Replaces the first subsequence of the input sequence that matches the pattern with the given replacement string.

Demo : Matcher class matches method

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegularExpressionDemo2 {
   public static void main(String[] args) {
   ……
   Matcher matcher = pattern.matcher("Hello");
   boolean result = false;

  //use of matches() method
   result = matcher.matches();
   if(result)
      System.out.println("match() : Match found");
   else
      System.out.println("match() : No match found");
```

Demo : Matcher class matches method….

```
//Use of pattern
Pattern pattern = Pattern.compile("Hello");

 matcher = pattern.matcher("Welcome");
    result = matcher.matches();
    if(result)
            System.out.println("match() : Match found");
     else
            System.out.println("match() : No match found");
      }
```

Demo : Matcher class find method

- Sample code given here

```
//use of find() method
    Pattern pattern = Pattern.compile("Java");
    Matcher matcher = pattern.matcher("Welcome to Java");
    boolean result = false;
//looking for "Java" in "Welcome to Java"
     result = matcher.find();
     if(result)
              System.out.println("find() : Match found");
     else
              System.out.println("find() : No match found");
```

Demo : Matcher class find method

- Output:
- find() : Match found
- find() : No match found

```
pattern = Pattern.compile("Welcome to Java");
matcher = pattern.matcher("Java");
result = matcher.find();
//looking for "Welcome to Java" in "Java"
 if(result)
       System.out.println("find() : Match found");
 else
       System.out.println("find() : No match found");
```

Demo : Matcher class find & start method

- Output:
- abc is found at 0
- abc is found at 8
- abc is found at 12
- abc is found at 20

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegularExpressionDemo4 {
 public static void main(String[] args) {
    Pattern pattern = Pattern.compile("abc");
    Matcher matcher = pattern.matcher("abc xyz abc abc xyz abc xyz");
    while(matcher.find()){
    System.out.println("abc is found at " + matcher.start());
   }}
}
```

Demo : Matcher class find & end method

- Output:
- abc is found at 3
- abc is found at 11
- abc is found at 15
- abc is found at 23

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegularExpressionDemo5 {
public static void main(String[] args) {
    Pattern pattern = Pattern.compile("abc");
   Matcher matcher = pattern.matcher("abc xyz abc abc xyz abc xyz");
   while(matcher.find()) {
    System.out.println("abc is found at " + matcher.end());
    }
}
}
```

The group() method

- The group method is used to search for a certain group of characters.
- This is used to treat multiple character in a group as a single unit.
- Quantifiers will be used to specify occurrence in a group.
- Quantifiers are as below.

Quantifier	Description
+	Match one or more.
*	Match zero or more.
?	Match zero or one.

- E.g. a+ will match for a string as “a”, “aa”, “aaa” etc.

Demo: group method

- **Output:**
- **Match found : A**
- **Match found : AA**
- **Match found : AAA**
- **Match found : AAAA**

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class RegularExpressionDemo5 {
public static void main(String[] args) {

   Pattern pattern = Pattern.compile("A+");
  Matcher matcher = pattern.matcher("A AA AAA AAAA");

    while(matcher.find()){
   System.out.println("Match found : " + matcher.group());
                }
                }
}
```

Demo: group method

- Sample code

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegularExpressionDemo6 {
public static void main(String[] args) {
  Pattern pattern = Pattern.compile("A+");
  Matcher matcher = pattern.matcher("B BB BBB BBBB");
  while(matcher.find()){
    System.out.println("Match found : " + matcher.group());
     }         }
}
```

Common matching symbols

Symbol	Description
.	Matches any character
^regex	Finds regex that must match at the beginning of the line.
regex$	Finds regex that must match at the end of the line.
[abc]	Set definition, can match the letter a or b or c.
[abc][vz]	Set definition, can match a or b or c followed by either v or z.
[^abc]	When a caret appears as the first character inside square brackets, it negates the pattern. This pattern matches any character except a or b or c.
[a-d1-7]	Ranges: matches a letter between a and d and figures from 1 to 7, but not d1.
X|Z	Finds X or Z.
XZ	Finds X directly followed by Z.
$	Checks if a line end follows.

Common matching symbols

```
//checking all characters
String s = "abc";
System.out.println(s.matches("..."));
System.out.println(s.matches(".."));

//checking true or True
s = "true";
System.out.println(s.matches("[tT]rue")
);
s = "True";
System.out.println(s.matches("[tT]rue")
);
```

Prints
true
false

Prints
true
true

```
//checking true/True or false/False
s = "true";
System.out.println(s.matches("[tT]rue|[fF]alse")
);
s = "false";
System.out.println(s.matches("[tT]rue|[fF]alse")
);

//checks if string is containing 3 letters
s = "Cat";
System.out.println(s.matches("[a-zA-Z]{3}"));
s = "Java";
System.out.println(s.matches("[a-zA-Z]{3}"));
```

Prints
true
true

Prints
true
false

Meta characters

Meta Character	Description
\d	Any digit, short for [0-9]
\D	A non-digit, short for [^0-9]
\s	A whitespace character, short for [\t\n\x0b\r\f]
\S	A non-whitespace character, short for [^\s]
\w	A word character, short for [a-zA-Z_0-9]
\W	A non-word character [^\w]
\S+	Several non-whitespace characters
\b	Matches a word boundary where a word character is [a-zA-Z0-9_].

Meta characters

- Sample Code


```
//checks no digits are at the beginning
s = "Hello";
System.out.println(s.matches("^[^\\d].*"));
s= "123Hello";
System.out.println(s.matches("^[^\\d].*"));
```


```
//checks if the word is not containing character b
s = "abc";
System.out.println(s.matches("([\\w&&[^b]])*"));
s = "1ack";
System.out.println(s.matches("([\\w&&[^b]])*"));
```

Summary :

With this we have come to an end of our session, where
we discussed about ….

- What are Regular Expressions?
- Need of Regular Expressions?
- Package and the classes used to implement the same.
- The methods those are available for implementing Regular Expressions.

At the end of this session, we see that you are now able to:

- Develop applications using Regular Expressions.

Appendix

References

Thank you

Reference Material : Websites & Blogs

- http://www.vogella.com/tutorials/JavaRegularExpressions/article.html
- http://www.tutorialspoint.com/java/java_regular_expressions.htm
- https://docs.oracle.com/javase/tutorial/essential/regex/

Reference Material : Books

- ***Head First Java***
  - *By: Kathy Sierra, Bert Bates*
  - *Publisher: O'Reilly Media, Inc.*

- ***Java Complete Reference***
  - *By Herbert Schildt*

Persistent

Thank you!

Persistent University